1

ARMADA Middleware Suite

T. Abdelzaher, S. Dawson, W. Feng, S. Ghosh†, F. Jahanian,
S. Johnson, A. Mehra, T. Mitton, J. Norton†, A. Shaikh,
K. Shin, V. Vaidyan†, Z. Wang, H. Zou

Real-Time Computing Laboratory
Dept. of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122

† Honeywell Technology Center
Honeywell, Inc.
3660 Technology Drive
Minneapolis, MN 55418

Abstract

ARMADA is a set of communication and middleware services that provide support for fault-tolerance and end-to-end guarantees for embedded real-time distributed applications. Since the real-time performance of such applications depends heavily on the communication subsystem, the first goal of the project was to develop a predictable communication service and architecture to ensure QoS-sensitive message delivery. Second, ARMADA aimed to offload the complexity of developing fault-tolerant applications from the application programmer by focusing on a collection of modular, composable middleware for fault-tolerant group communication and replication under timing constraints. Finally, we developed tools for testing and validating the behavior of our services. In this paper, we give a brief overview of the ARMADA project, describing the architecture and services it provides, along with its implementation status.

Keywords: distributed real-time system, communication protocol, fault-tolerant applications.

1 Introduction

Real-time embedded systems have evolved during the past several decades from small custom-designed digital hardware to large distributed processing systems. As these systems become more complex, their interoperability, evolvability and cost-effectiveness requirements motivate the use of commercial-off-the-shelf (COTS) components. This introduces the challenge of constructing dependable and predictable real-time services for application developers on top of inexpensive hardware and software components which have minimal support for timeliness and dependability guarantees. We are addressing this challenge in the ARMADA project.

ARMADA is a collaborative project between the Real-Time Computing Laboratory (RTCL) at the University of Michigan and the Honeywell Technology Center. The goal of the project is to develop and demonstrate an integrated set of communication and middleware services and tools which realize embedded fault-tolerant and real-time services on distributed, evolving computing platforms. These techniques and tools together compose an environment of capabilities for designing, implementing, modifying, and integrating real-time distributed systems. Key challenges addressed by the ARMADA project include: timely delivery of services with end-to-end hard and soft real-time constraints; dependability of services in the presence of hardware or software failures; scalability of computation and communication resources; and exploitation of open systems and emerging standards in operating systems and communication services. Due to space considerations, this paper provides an introduction and basic overview of the different middleware services developed as part of the project. For a more complete description of the ARMADA project and its ongoing research, the reader is referred to [1].

The ARMADA project was divided into three complementary thrust areas: (i) low-level middleware for real-time communication support, (ii) middleware services for fault-tolerant group communication and replication, and (iii) dependability evaluation and validation tools. Figure 1 summarizes the structuring of the ARMADA environment.

The first thrust focused on the design and development of real-time communication services for a microkernel. These services could then be used as a foundation for constructing higher-level real-time middleware services. We present a generic architecture for designing the communication subsystem which allows predictability and QoS

*This work is supported in part by a research grant from the Defense Advanced Research Projects Agency, monitored by the U.S. Air Force Rome Laboratory under Grant F30602-95-1-0044.

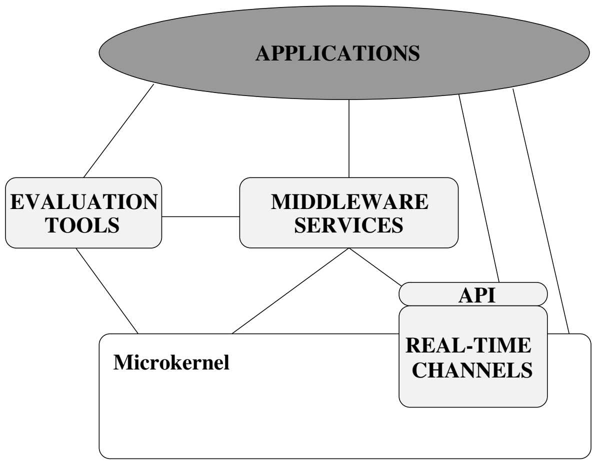


Figure 1: Overview of ARMADA Environment.

guarantees to be maintained. This architecture is independent of any particular communication service. We illustrate the architecture by presenting the design of the *real-time channel*: a low-level communication service that implements a simplex, ordered virtual connection between two networked hosts and provides either deterministic or statistical end-to-end delay guarantees between each sender-receiver pair.

The second thrust of the project focused on a collection of modular and composable middleware services (or building blocks) for constructing embedded applications. A layered open-architecture supports modular insertion of a new service or implementation as requirements evolve over the lifespan of a system. The ARMADA middleware services include a suite of fault-tolerant group communication services with real-time guarantees, called RTCAST, which are intended to support embedded applications with fault-tolerance and timeliness requirements. ARMADA also includes a real-time primary-backup (RTPB) replication service, which ensures *temporally consistent* replicated objects on redundant nodes.

The third thrust of the project was to build a toolset for validating and evaluating the timeliness and fault-tolerance capabilities of the target system. Although a more complete description of the tools is beyond the scope of this paper, details are available in [1, 2].

Figure 2 gives an overview of a prospective application to illustrate the utility of our services for embedded real-time fault-tolerant systems. We believe that by developing our middleware services in conjunction with a real-world application it is possible to design services and, just as importantly, interfaces to those services, that are robust and easily usable by application developers. This application, developed at Honeywell, is a subset of a command and control facility. Consider a radar installation where a set of sensors are used to detect incoming threats. Hypotheses are formed regarding the identity and positions of the threats, and their flight trajectories are computed accordingly. These trajectories are extrapolated into the future and deadlines are imposed to intercept them. The time intervals during which the estimated threat trajectories are reachable from various ground defense bases are estimated, and appropriate resources (weapons) are committed to handle the threats. Eventually, the weapons are released to intercept the threats.

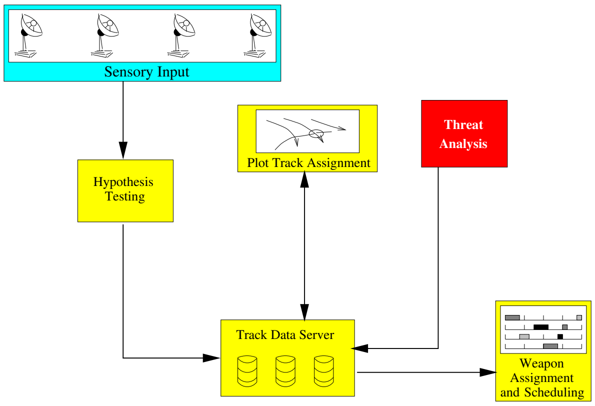


Figure 2: A command and control application

The services required to support applications such as this are derived naturally from their operating requirements. In our sample application for example, based on the anticipated system load, communication between different system components (the different boxes in Figure 2) must occur in bounded time to ensure a bounded end-to-end response from threat detection to weapon release. Our real-time communication service computes and enforces predictable deterministic bounds on message delays given an application traffic specification. Critical system components such as hypothesis testing and threat identification have high dependability requirements which are best met using active replication. For such components, RTCAST exports multicast and membership primitives to facilitate fault detection, fault handling, and consistency management of actively replicated tasks. Similarly, extrapolated trajectories of identified threats represent critical system state. A backup of such state needs to be maintained continually and updated to represent the current state within a tolerable consistency (or error) margin. The RTPB service is implemented to meet such requirements.

The remainder of this paper describes these services in more detail. Section 2 introduces our generic architecture for designing the communication subsystem. Section

3 presents a description of the middleware services. Finally, section 4 provides a description of our implementation platform and a summary of the current implementation status of each project component.

2 ARMADA Real-Time Communication Architecture

In this section we present the overall architecture of the ARMADA real-time communication service. This service provides a number of real-time communication guarantees which can be utilized by other middleware to provide higher-level services. The first half of this section describes the goals and paradigm of our real-time communication architecture. The second part highlights key aspects of the service architecture that meet our stated goals.

2.1 Real-Time Communication Service: Goals and Paradigm

The primary goal of the real-time communication service is to provide applications and middleware with a service that can be used to request and obtain (real-time) guaranteed QoS unicast connections between hosts. Such a service must satisfy three primary architectural requirements for guaranteed-QoS communication [3]: (i) maintenance of per-connection QoS guarantees, (ii) overload protection via per-connection traffic enforcement, and (iii) fairness to best-effort traffic. In order for the service to satisfy these requirements, an application must specify the worst-case traffic profile it expects to generate, and the end-to-end QoS it desires for parameters such as delay, bandwidth, and likelihood of packet loss. A secondary, but also very important, goal is to design the service in a way that allows (a) constituent architectural components to be reused for other middleware services, and (b) flexibility to realize other QoS paradigms and service models. This is important because in our environment, multiple middleware services must coexist and interoperate; reusing architectural components, whenever possible, makes service integration relatively easier. Equally important, in order to realize generic components, one must decouple the service model from the component architecture. This facilitates the ability to extend the service to more relaxed QoS models such as probabilistic guarantees, QoS negotiation, and adaptation.

To realize the real-time communication service, we adopt the service model of *real-time channels* [4, 5], a paradigm for guaranteed-QoS communication in packet-switched networks. This model is similar to other proposals for guaranteed-QoS communication [6], and we have developed significant insights by extending the model appropriately for practical use [3, 7]. A real-time channel is a simplex, fixed-route, virtual connection between a source and a destination host, with sequenced messages and associated performance guarantees on message delivery.

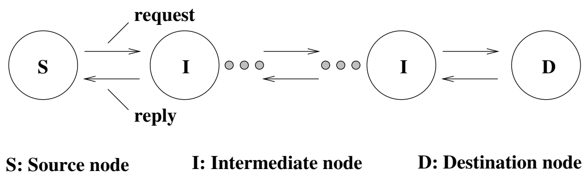


Figure 3: Signaling between two hosts.

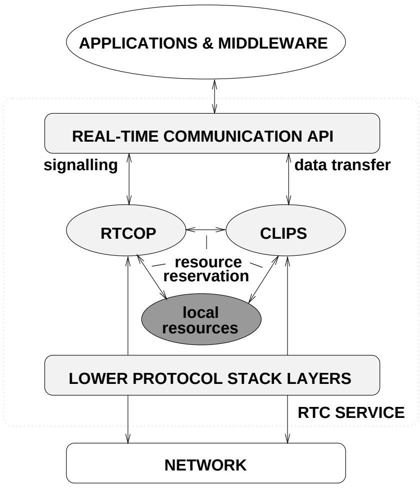


Figure 4: Software architecture for real-time communication.

Real-time communication via real-time channels is performed in three phases (see Figure 3). In the first phase, the source host S (sender) creates a channel to the destination host D (receiver) by specifying the channel's traffic parameters and QoS requirements. Signaling requests are sent from S to D via one or more intermediate (I) nodes; replies are delivered in the reverse direction from D to S. At each hop, an admission test is performed and sufficient resources are allocated and reserved to meet the channel's QoS requirements. If the channel is successfully established, the second phase begins and S uses the channel to send messages to D. Finally, the sender tears down the cahnnel when communication is complete so that resources may be released. This is the third phase of the communication.

2.2 Service Architecture

Figure 4 illustrates the software architecture of our guaranteed-QoS service at hosts and intermediate nodes. The core functionality of the service is realized via three distinct components that interact to provide guaranteed-QoS communication: the real-time communication API, the real-time channel ordination protocol (RTCOP), and the communication library for implementing priority semantics (CLIPS). Applications access the service via the RTC API. End-to-end signalling and resource reservation is coordinated by RTCOP and run-time management of resources for data transfer is performed by CLIPS. The CORDS services framework, described in section 4, provides a common interface which allows these components to be implemented independently and composed into a single protocol stack. As mentioned above, the run-time resource management in the service architecture is based in large part on the architecture proposed in [3], with significant enhancements to accommodate the specific requirements of the ARMADA project and its implementation environment. Details on the internals of the service components and their interaction are provided in [8]. A brief description of the implementation status can be found in Section 4.

3 ARMADA Composable Middleware

The previous section introduced the architecture of the ARMADA real-time communication service. This section now presents a collection of modular and composable middleware services which build upon this architecture to provide support for constructing embedded applications. The ARMADA middleware can be divided into two relatively independent suites of services:

- RTCAST group communication services, and
- RTPB real-time primary-back replication service.

The remainder of this section describes each of these services in more detail, and section 4 presents the current status of their implementation.

3.1 RTCAST Group Communication Services

Process groups are a widely-studied paradigm for designing dependable distributed systems in both asynchronous [9, 10, 11, 12] and synchronous [13, 14, 15] environments. In this approach, a distributed system is structured as a group of cooperating processes which provide services to an application. A process group may be used, for example, to provide active replication of system state or to rapidly disseminate information from an application to the members of the group. Two key primitives for supporting process groups in a distributed environment are *fault-tolerant multicast communication* and *group membership*. Coordination of a process group must address several subtle issues including delivering messages to the group in a reliable fashion, maintaining consistent views of group membership, and detecting and handling process or communication failures. If multicast messages are atomic and globally ordered, process group members can be kept consistent when process state is determined by initial state and the sequence of received messages.

We developed RTCAST to provide a lightweight fault-tolerant multicast and membership service for real-time process groups which exchange periodic and/or aperiodic messages. The RTCAST group communication does not require acknowledgments for every message, and message delivery is immediate without needing additional "rounds" of message transmissions. RTCAST is designed to support hard real-time guarantees without requiring a static schedule to be computed *a priori* for application tasks and messages. Instead, an on-line schedulability analysis component performs admission control on multicast messages. The proposed multicast and membership protocols are part of a larger suite of middleware group communication services that form a composable architecture for the development of embedded real-time applications. As illustrated in Figure 5, the RTCAST suite of services include a timed atomic multicast, a group membership service, and an admission control service. The first two are tightly coupled and thus are considered a single service. Clock synchronization is typically required for real-time protocols and is provided by the clock synchronization service. To support portability, a *virtual network interface* layer exports a uniform network abstraction. This interface transparently handles different network topologies, each potentially having different connectivity, timing, or bandwidth characteristics, by exporting a generic network abstraction to upper layers. The network is only assumed to support a unicast datagram service. Finally, the top layer provides an application programming interface for implementing distributed applications using real-time process groups.

Note that each component of the RTCAST exports a well known interface to the other software components, but each can be implemented independently and the system designer can choose to compose only those components needed to achieve the desired level of service guarantee. The system designer might also select between different versions of each component, depending upon the particular requirements of the application. For example, in a system with hard real-time constraints, an admission control algorithm might be used which employed a strict, worst-case analysis when allocating resources and admitting message traffic, while in a system with softer constraints a different implementation of the admission control protocol could be inserted that would provide higher throughput or faster response time at the expense of having a less predictable probability of missed deadlines. If the system designer only needs RTCAST's membership and total ordering services, he or she may even choose not to include the admission control protocol at all, reducing the size and system resources required by the service.

RTCAST supports bounded-time message transport, atomicity, and total order for multicasts within a group of communicating processes in the presence of processor crashes and communication failures. It guarantees agreement on membership among the communicating proces-

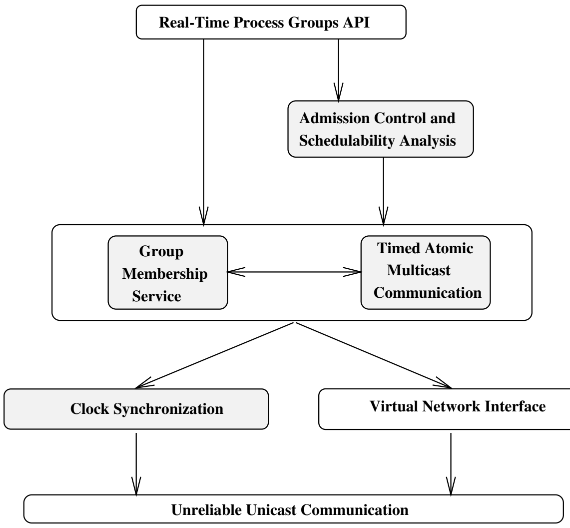


Figure 5: Software architecture for the RTCAST middleware services.

sors, and ensures that membership changes are atomic and ordered with respect to multicast messages. It is designed as a a logical ring, with senders taking turns multicasting messages over the network. A processor's turn comes when the logical token arrives, or when it times out waiting for it. After its last message, each sender multicasts a heartbeat that is used for crash detection. The heartbeat received from an immediate predecessor also serves as the logical token. Destinations detect missed messages using sequence numbers and when a processor detects a receive omission, it crashes. Each processor, when its turn comes, checks for missing heartbeats and eliminates the crashed members, if any, from the group membership by multicasting a membership change message. Although this gap detection policy may seem excessively restrictive, in most fault-tolerant systems the physical network is redundant, and the likelihood of transmission failures is low. Furthermore, by removing incorrect processors from the group immediately, we can provide immediate delivery at all correct nodes, which is a significant performance advantage.

A more detailed explanation of RTCAST's operation and proof of the correctness of its algorithms are available in [16].

3.2 Real-Time Primary-backup (RTPB) Replication Service

While the RTCAST group communication services are a suitable mechanism for implementing active replication in real-time systems, some applications may not need such a strong level of redundancy; instead, they may prefer to make use of the primary-backup approach. Unfortunately, traditional primary backup systems can require extra management overhead and a significant amount of communication resources as changes to the application state are propagated from the primary to the backup. The timing uncertainty involved in this approach may make it unsuitable for use in hard real-time systems. Our Real-Time Primary Backup (RTPB) service solves this problem by exploiting application data semantics and allowing the backup to maintain a less current copy of the data that resides on the primary [17].

With sufficiently recent data, the backup can safely supplant a failed primary; the backup can then reconstruct a consistent system state by extrapolating from previous values and new sensor readings. However, the system must ensure that the lag between the primary and the backup data is bounded within a predefined time window. Data objects may have distinct tolerances in how far the backup can lag behind the primary before the object state becomes stale. The challenge is to bound the distance between the primary and the backup such that consistency is not compromised, while minimizing the overhead of exchanging messages between the primary and its backup. We have designed RTPB to meet these goals.

RTPB employs two types of consistency checking to ensure that data at the backup is acceptably recent with respect to the primary. One is the *external temporal consistency* between an object in the external world and its image in the real-time application, and the other is the *inter-object temporal consistency* between different objects or events.

External temporal consistency is satisfied for an object i if the timestamp of i at a server is no later than a predetermined time from its timestamp at the client in the external world. In other words, the state of the primary server must closely reflect that of the actual world. This consistency is also needed at the backup if the backup is to successfully replace the primary when the primary fails. The consistency restriction placed on the backup may not be as tight as that on the primary but must be within a tolerable range for the intended applications.

Inter-object temporal consistency is defined as the time lag allowed between when an update to one object is performed and when an update must be completed for a second object. This lag may be different on the primary and the backup. For example, an application may specify that the replica of object B on the primary must be updated within 100 ms after an update to the replica of object A on the primary, but the replica of object B at the backup need only be updated within *200* ms after an update to the replica of object A at the backup.

By specifying appropriate values for the desired external and inter-object consistency constraints, a system designer can use RTPB to meet desired consistency semantics while incurring less overhead than with traditional replication techniques. In order to ensure these guarantees, RTPB operates as follows. Both the client and the primary server run on the same physical node. First, the client must register any object that is to be replicated with the RTPB primary server, specifying both the update period and the temporal bounds that must be observed at both the primary and the backup. The primary server performs admission control to decide whether it has the resources to update the object at the requested frequency. During registration, any necessary resources for the object are reserved on both the primary and backup servers. If there are not enough resources available to satisfy the request, it is rejected.

Once an object is admitted to the system, it will be backed up by the primary server periodically, always ensuring that the backup is within the requested temporal bound of the primary. Note that object backups are decoupled from the application; that is, updates between the application and the primary occur independently from updates between the primary and the backup. This reduces the latency seen by the application when updating objects while still maintaining the requested consistency semantics.

The primary and backup servers periodically exchange heartbeat messages. If the backup fails, a new one is started by the primary. If the primary fails, the backup takes over as the primary and starts a new copy of the application using the replicated data objects. It also initiates a new backup server to maintain redundancy.

A brief description of the implementation status of RTPB is given in section 4 of this paper. For a more detailed description of the operation and architecture of the Real-Time primary backup replication service, as well as proofs of the algorithm's correctness, see [17, 18].

4 ARMADA Middleware Services Implementation

To facilitate the development of communication oriented services, our communication subsystem is implemented using the x-kernel object-oriented networking framework originally developed at the University of Arizona [19], with the CORDS framework extensions for controlled allocation of system resources [20]. The advantage of using the x-kernel is the ease of composing protocol stacks. An x-kernel communication subsystem is implemented as a configurable graph of protocol objects. It allows easy reconfiguration of the protocol stack by adding or removing protocols. The CORDS framework runs under the OSF MK7.2 Mach operating system, on a testbed of Intel Pentium-based PCs connected by an off-the-shelf 10 Mbps ethernet network.

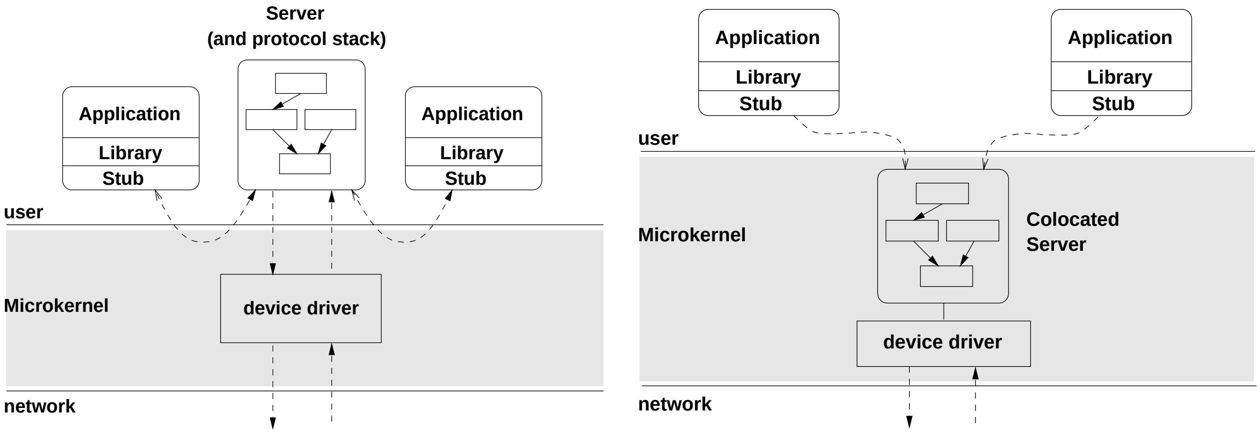
Following the microkernel philosophy, our services were first implemented as user level servers. Clients of the services are separate processes that communicate with the servers via a user library which exports the desired middleware API. Communication-oriented services generally implement their own protocol stack which lies on top of the kernel-level communication driver. Although the protocol stacks usually run in user space, because we are implementing them on a micro-kernel platform we are able to easily migrate them into the operating system kernel to improve performance and reduce overhead due to context switches and data copying. By running in the kernel, we also benefit from a higher scheduling priority which results in more predictable thread scheduling and processor utilization. Figure 6-a and 6-b illustrate the configurations of user level servers and servers which are colocated in the kernel, respectively.

Whether the server runs in user space or is colocated in the microkernel, processes use the same service API to communicate with it. Automatically-generated stubs interface the user library (implementing the service API) to the microkernel and the server process. These stubs hide the details of the kernel's local communication mechanism from the programmer of the real-time service, thus making the service code independent of the specifics of the underlying microkernel.

All of the ARMADA middleware services have been implemented and tested under the CORDS framework. The Real-Time Channels implementation makes use of the services provided by CORDS to reserve resources on a per-connection basis. It also modifies the lower-level ethernet drivers to support link scheduling and resource reclamation. Real-Time channels currently runs as a user-level server for development purposes, but as we have already mentioned it can easily be moved into the operating system kernel to improve performance. The implementation includes an API which exports the Real-Time Channels service interface to user-level applications.

RTCAST has also been fully implemented within the CORDS framework. It currently runs as part of the operating system kernel, which gives it a very fast response time and greatly reduces the latencies associated with fault-detection and message processing. We have designed a robust API and an additional service library for RTCAST, and are in the process of implementing these APIs for use by application developers. We have conducted extensive performance tests of RTCAST, and the results compare favorably to other group multicast protocols [21].

The Real-time Primary Backup system has been implemented and tested with a sample application. It currently runs as a user-level server, though it also can be moved into the kernel to improve performance. An API has been implemented that enables applications to specify their object consistency requirements and replicate those objects on the backup. A performance evaluation of this service is available in [18].



(a) User level server configuration (b) Colocated server

Figure 6: Service implementation.

In the near future, we plan to port these services to the Windows NT environment, taking advantage of its microkernel architecture and wide availability to make the ARMADA middleware services more accessible to real-time system developers. Although Windows NT currently does not provide OS-level real-time support, this functionality is currently being developed by a number of third parties. The CORDS framework which we use for our services implementations has been ported to NT, and we have been made aware of a number of other third-party extensions which are also being developed for this purpose. We therefore feel that it is a promising platform for future real-time communication development.

5 Conclusions

This paper presented the architecture of the ARMADA project conducted at the University of Michigan in collaboration with the Honeywell Technology Center. We described a number of communication and middleware services developed in the context of this project, and illustrated the general methodology adopted to design and integrate these services. For modularity and composability, ARMADA middleware was realized as a set of servers on top of a microkernel-based operating system. Special attention was given to the communication subsystem since it is a common resource for the middleware services developed. We proposed a general architecture for QoS sensitive communication, and also described a communication service that implements this architecture.

We are currently redesigning an existing command and control application to benefit from the ARMADA middleware. The application requires bounded-time end-to-end communication delays which are guaranteed by our communication subsystem, as well as fault-tolerant replication and backup services which are provided by RTCAST's group communication and membership support. It also makes use of the primary-backup replication service to ensure critical data integrity.

Our services and tools are designed independently of the underlying microkernel or communication subsystem; our choice of experimentation platform was based largely on the rich protocol development environment provided by *x*-kernel and CORDS. For better portability, we are extending our communication subsystem to provide a socket-like API and looking at implementing it on other platforms, such as Windows NT. We are also investigating the scalability of the services developed. We are looking into appropriate ways of defining generic structural system components and composing large architectures from these components such that desirable properties are globally preserved. It is our hope that by developing the "tokens" and "operators" of such system compositions, we will be able to build predictable analytical and semantic models of larger systems based on the properties of their individual constituents.

References

[1] T. Abdelzaher, S. Dawson, W. Feng, F. Jahanian, S. Johnson, A. Mehra, T. Mitton, A. Shaikh, K. Shin, Z. Wang, and H. Zou, "ARMADA Middleware and Communication Services," Technical report, Dept. of Electrical Engineering and Computer Science, University of Michigan, 1997. Submitted for publication.

[2] S. Dawson, F. Jahanian, and T. Mitton, "Testing of Fault-Tolerant and Real-Time Distributed Systems via Protocol Fault Injection," in *International Symposium on Fault-Tolerant Computing*, pp. 404–414, Sendai, Japan, June 1996.

[3] A. Mehra, A. Indiresan, and K. Shin, "Structuring Communication Software for Quality of Service Guarantees," in *Proc. 17th Real-Time Systems Symposium*, pp. 144–154, December 1996.

[4] D. Ferrari and D. Verma, "A Scheme for Real-time Channel Establishment in Wide-Area Networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 3, pp. 368–379, April 1990.

[5] D. Kandlur, K. Shin, and D. Ferrari, "Real-time Communication in Multi-hop Networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 5, no. 10, pp. 1044–1056, October 1994.

[6] C. Aras, J. Kurose, D. Reeves, and H. Schulzrinne, "Real-Time Communication in Packet-Switched Networks," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 122–139, January 1994.

[7] A. Mehra, A. Indiresan, and K. Shin, "Resource Management for Real-Time Communication: Making Theory Meet Practice," in *Proc. 2nd Real-Time Technology and Applications Symposium*, pp. 130–138, June 1996.

[8] A. Mehra, A. Shaikh, T. Abdelzaher, Z. Wang, and K. Shin, "Realizing Guaranteed-QoS Communication Services on a Micro-kernel Operating System," In preparation, July 1997.

[9] K. Birman, "The Process Group Approach to Reliable Distributed Computing," *Communications of the ACM*, vol. 36, no. 12, pp. 37–53, December 1993.

[10] Y. Amir, D. Dolev, S. Kramer, and D. Malki, "The Transis Approach to High Availability Cluster Communication," *Communications of the ACM*, vol. 39, no. 4, pp. 64–70, April 1996.

[11] R. van Renesse, K. Birman, and S. Maffeis, "Horus: A Flexible Group Communication System," *Communications of the ACM*, vol. 39, no. 4, pp. 76–83, April 1996.

[12] S. Mishra, L. Peterson, and R. Schlichting, "Consul: A Communication Substrate for Fault-tolerant Distributed Programs," *Distributed Systems Engineering Journal*, vol. 1, no. 2, pp. 87–103, December 1993.

[13] H. Kopetz and G. Grünsteidl, "TTP – A Protocol for Fault-Tolerant Real-Time Systems," *IEEE Computer*, vol. 27, no. 1, pp. 14–23, January 1994.

[14] Y. Amir, L. Moser, P. Melliar-Smith, D. Agarwal, and P. Ciarfella, "The Totem Single-Ring Ordering and Membership Protocol," *ACM Transactions on Computer Systems*, vol. 13, no. 4, pp. 311–342, November 1995.

[15] F. Cristian, B. Dancy, and J. Dehn, "Fault-Tolerance in the Advanced Automation System," in *Proc. of Fault-Tolerant Computing Symposium*, pp. 6–17, June 1990.

[16] T. Abdelzaher, A. Shaikh, F. Jahanian, and K. Shin, "RTCAST: Lightweight Multicast for Real-Time Process Groups," in *Proc. IEEE Real-Time Technology and Applications Symposium (RTAS '96)*, pp. 250–259, Boston, MA, June 1996.

[17] A. Mehra, J. Rexford, and F. Jahanian, "Design and Evaluation of a Window-Consistent Replication Service," *IEEE Transactions on Computer*, vol. 46, no. 9, , September 1997.

[18] H. Zou and F. Jahanian, "Real-Time Primary Backup Replication with Temporal Consistency," Technical report, Dept. of Electrical Engineering and Computer Science, University of Michigan, 1997.

[19] N. Hutchinson and L. Peterson, "The *x*-Kernel: An Architecture for Implementing Network Protocols," *IEEE Trans. Software Engineering*, vol. 17, no. 1, pp. 1–13, January 1991.

[20] F. Travostino, E. Menze, and F. Reynolds, "Paths: Programming with System Resources in Support of Real-Time Distributed Applications," in *Proc. IEEE Workshop on Object-Oriented Real-Time Dependable Systems*, February 1996.

[21] T. Abdelzaher, A. Shaikh, S. Johnson, F. Jahanian, and K. Shin, "RTCAST: Lightweight Multicast for Real-Time Process Groups," Technical report, Dept. of Electrical Engineering and Computer Science, University of Michigan, 1997. Submitted for publication.